Distribution of solutions of FreeCell (Algebraic System, Logic, Language and Related Areas in Computer Science)

AUTHOR(S):

JIMBO, Shuji; Yamada, Kotaro; Imamura, Sho; Ueno, Kazuki; Okada, Hiromu

CITATION:

JIMBO, Shuji ...[et al]. Distribution of solutions of FreeCell (Algebraic System, Logic, Language and Related Areas in Computer Science). 数理解析研究所講究録 2019, 2130: 85-90

ISSUE DATE:

2019-10

URL:

http://hdl.handle.net/2433/254751

Distribution of solutions of FreeCell

Shuji JIMBO*, Kotaro Yamada,
Sho Imamura, Kazuki Ueno, and Hiromu Okada
Okayama University

*jimbo-s@okayama-u.ac.jp

Abstract

There is a plan by the authors for developing a FreeCell solver by supervised deep learning that searches very few positions as compared with solvers currently released. A large number of solutions of high quality to initial positions in FreeCell are required for the supervised deep learning. A FreeCell solver without deep learning is being developed to make the data sets for deep learning. In this paper, construction of the latter FreeCell solver and the distribution of solutions to initial positions of FreeCell obtained by experiments using the solver are described.

Keywords. deep learning, A* algorithm, search algorithm, FreeCell.

1 Introduction

FreeCell is a solitaire card game that is played on Windows OS for a long time. It is fairly easy to find a solution of a given initial position, but it is generally hard to find the optimum moves to solve the initial position. Actually, if an initial position is picked according to the uniform distribution, then it is easy to find a solution of the initial position of length less than or equal to 120 in a couple of seconds with high probability using a FreeCell solver that we are currently developing. A small minority of them are difficult to solve. Furthermore, there may be an unsolvable initial position in several tens of thousands of randomly selected ones. For example, MS FreeCell No. 11982 is a famous unsolvable initial position with the smallest problem number.

After the current century started, excellent FreeCell solvers were realised[4][1][2]. Paul et al developed a FreeCell solver that efficiently finds an optimum solution to a given initial position[5]. Their solver will be described in Section 3. A part of their idea on improving a heuristic function in A* algorithm is used in our solver.

We are currently developing a FreeCell solver that can find a solution close to an optimum one. We have another plan to develop a FreeCell solver that searches as few positions as possible by using a neural network (NN) as an important component. The quality of NN depends on the data sets for training. The solver being currently developed are going to be used to make data sets of high quality for the deep learning in the plan. The fundamental structure of the neural network in the plan is a one dimensional convolutional NN with the ResNet structure.

In the following sections, construction of the FreeCell solver currently developed and the distribution of solutions to initial positions of FreeCell obtained by experiments using the solver are described.

2 Solitaire Card Game FreeCell

The rule of FreeCell is as follows.

Initially, 52 cards are randomly dealt into eight piles, where the first four piles are of length six and the rest are of length five. The objective of the game is to move all cards onto four different piles, called the foundation. For each suit, there is a foundation pile corresponding to it. Each foundation pile has to be piled up with the cards of the suit from the ace at the bottom to the king at the top. Additionally, there are initially empty four cells, called free cells

A legal move consists of picking a card and dropping it. Only a card on a top of a tableau pile or in a free cell can be picked. The picked card can be dropped on an empty free cell, on top of foundation piles if all lower rank cards of the same suit are already there, and, on top of tableau piles if the top card of the pile is of one higher rank and of opposite color. For example, the legal moves in the position in Fig. 1 are the following twelve moves: H9 on S10, D6 on D5, S5 on D6, S5 on S4, HQ into a free cell, H9 into a free cell, H5 into a free cell, CQ into a free cell, D6 into a free cell, S10 into a free cell, S5 into a free cell, and C9 into a free cell.

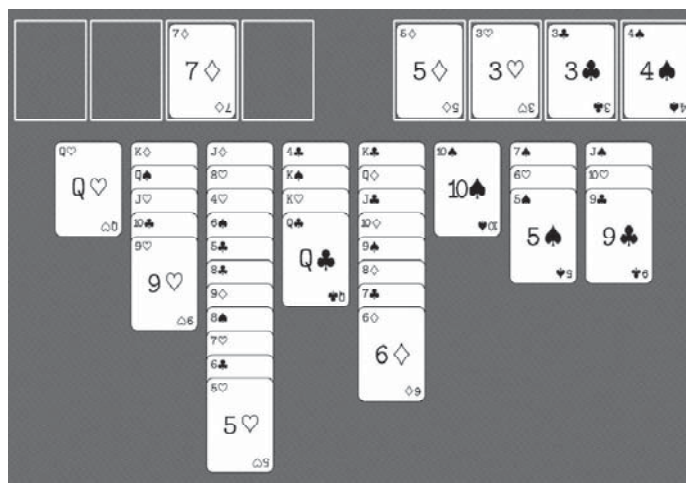

Figure 1: Legal moves in an position of FreeCell

For each positive integer n, the initial position corresponding to n in Microsoft FreeCell is made from pseudo random numbers generated by Microsoft software. It is well known that very rare initial positions have no solutions if initial positions are randomly selected from the uniform distribution. A solution of FreeCell is a list of moves from an initial position to the target or winning position in which all of the cards are in the foundation. We define the quality of a solution of FreeCell by the difference of the length of the solution from that of an optimum solution.

3 Earlier Studies

We can obtain an optimum solution by A* algorithm with an admissible heuristic function $h(n)$. A cost function $f(n)$ is defined as

$$f(n) = g(n) + h(n), \tag{1}$$

where $g(n)$ is the number of moves to the position n from the initial position. If the heuristic function $h(n)$ in Expression 1 is less than or equal to the least number of moves from n to a winning position, then the heuristic function $h(n)$ is called *admissible*, and an A* algorithm with an admissible heuristic function $h(n)$ always finds an optimum solution.

Let $N(n)$ denote the number of cards in the free cells or the tableau piles in the position n. Function $N(n)$ is a primitive admissible heuristic function. However, the performance of A* algorithm with the heuristic function $h(n) = N(n)$ fails the search for a large part of FreeCell positions due to its huge search space.

Paul et al has succeeded in finding optimum solutions for many initial positions by A* algorithm with the heuristic function $h(n) = N(n) + D(n)$, where $D(n)$ is the size of a minimum feedback blocking arc set of the deadlock graph of a FreeCell position[5]. Precisely speaking, they adopt a lesser heuristic function $D'(n) \leqq D(n)$ due to difficulty in computing the exact value of $D(n)$.

For a FreeCell position n, the deadlock graph $G(n)$ of n is a directed graph such that the vertex set of $G(n)$ is the set of 52 cards, and the arc set of $G(n)$ consists of two kind of arcs, that is to say, dependency arcs, and blocking arcs. Arc vw is a dependency arc if card v and w are of the same suit, and the rank of w is one less than the one of v. Arc vw is a blocking arc if card v and w are on an identical tableau pile, and card w is piled just on v, where card w is seen just under v on the screen of Microsoft FreeCell. All of the cycles in $G(n)$ have to vanish before winning. Since any dependency arc cannot be deleted, all of the arcs in a set B of blocking arcs such that every cycle in $G(n)$ includes an arc in B have to be deleted by moves. A move whose destination is not in the foundation is required for an arc in B to be deleted. Such a set B of blocking cards is called a *feedback blocking arc set* of a FreeCell position n. Value $D(n)$ is the minimum size of a feedback blocking arc set of n. A feedback blocking arc set of n of size $D(n)$ is called a minimum feedback blocking arc set of n. The heuristic function $h(n) = N(n) + D(n)$ is, therefore, admissible.

For example, Fig. 2 is the initial position of Microsoft FreeCell No. 1 with a minimum feedback blocking arc set, each arc of which is indicated by a downward arrow. The minimum feedback blocking arc set consists of thirteen arcs.

4 Construction of a FreeCell Solver

The current FreeCell solver to make wining moves for deep learning basically carries out the iterative deepening depth first search. It uses a large hash table to avoid searching identical positions. It can be altered behavior by giving command line parameters. Alternatives in the depth first search are selected randomly from the uniform distribution.

The algorithm uses a heuristic function $h(n)$ as the A* algorithm uses it. Function $h(n)$ is currently a linear combination of three component functions

$$h(n) = \alpha_1 h_1(n) + \alpha_2 h_2(n) + \alpha_3 h_3(n)\ , \tag{2}$$

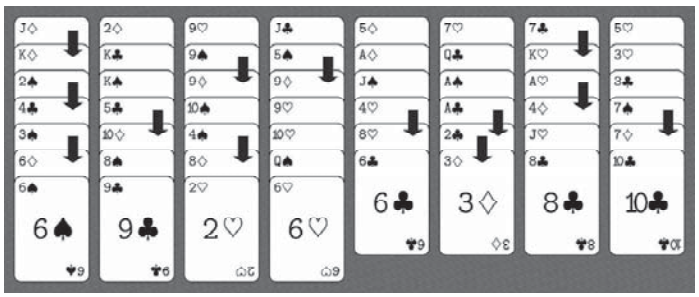

Figure 2: A minimum feedback blocking arc set of the initial position of Microsoft FreeCell No. 1

where $h_1(n)$ is the number of cards in free cells and tableau piles, $h_2(n)$ the number of one-suit blocking cards in tableau piles, and $h_3(n)$ is the number of cards that have not moved yet since the initial position. A card v in a tableau pile T is called a *one-suit blocking card* if there is a card w in T such that the rank of w is less than that of v and w is closer to the bottom of T than v.

The coefficients α_1, α_2, and α_3 in Expression 2 are currently determined by hand. The performance of the solver will be improved soon by adjusting the coefficients automatically by software. The notion of a one-suit blocking card is corresponding to a 1-suit cycle in a directed graph consisting of blocking edges and dependency edges that was introduced by Paul et al[5]. We call the directed graph a *deadlock graph*. We can have more accurate heuristic function $h'_2(n)$ than $h_2(n)$ by counting both 1-suit and 2-suit cycles in deadlock graphs. We expect that we can reduce the time to search shortest solutions greatly by using $h'_2(n)$ instead of $h_2(n)$.

5 Distribution of solutions of FreeCell

We performed experiments on distribution of solutions of FreeCell using a FreeCell solver used for making data sets for deep learning. The results are described in this section.

The number of initial positions of FreeCell used for the experiments is 400. Those initial positions were made from permutations of 52 cards randomly selected from the uniform distribution. The experiments were executed on a single 2.6 GHz Intel Xeon thread, where the size of an open addressing hash table was about 80 GiB.

First, 400 initial positions of FreeCell were classified into three classes according to difficulty of solving those initial positions as in Tab. 1. The class A of the hardest initial positions consists of the ones that can be solved only using all four free cells. The class B of moderately hard initial positions consists of the ones that can be solved using only three free cells, but cannot be solved if using three or more free cells is forbidden. The class C of the easiest initial positions consists of the ones that can be solved using only two free cells.

It can be seen from Tab. 1 that the ratio of the numbers of initial positions belong to the class A, B, and C to the total number of the initial positions, namely 400, are 0.75, 77.0, and 22.25, respectively. The number of the initial positions belonging to the class A from the first to the 64,000th games of Microsoft FreeCell is 419, and its ratio to the total

Table 1: The sizes of three classes of initial positions of FreeCell

A	B	C
3	308	89

number of the initial positions is about 0.655[3]D The lengths of the shortest solutions of the three initial positions in the class A are 81, 96, and 73, respectively. In the 400 initial positions used in the experiments, there is an initial position belonging to the class B that have a shortest solution of length 96. If only three free cells are permitted to be used, the length of a shortest solution to the initial position is 121.

We have had the following facts about the lengths of shortest solutions to the 400 initial positions used in the experiments. The length of a shortest solution is less than or equal to 100 for every initial position. There are at least 370 initial positions such that the length of a shortest solution is less than or equal to 90. There are at least 183 initial positions such that the length of a shortest solution is less than or equal to 80. We have a plan to find a shortest solution for every initial position in the 400 ones using an improved FreeCell solver.

6 Concluding Remarks

We are developing a FreeCell solver to make data sets for deep learning of ability to select moves and to evaluate positions. In this paper, an outline of the algorithm of the solver is described, then the distribution of solutions to initial positions randomly selected from the uniform distribution is shown. The solver basically executes the iterative deepening depth-first search algorithm. We have a plan for developing an A^* algorithm including neural networks trained by the deep learning. The analysis of distribution of the solutions to initial positions of FreeCell is quite rough because the number of samples is only 400.

We have a plan for improving the FreeCell solver so as to find shortest solutions for almost all initial positions in a short time. For instance, it is desired that a shortest solution can be found in a minute on an average by using a usual PC in an office. We expect that the quality of the data sets for the deep learning will be greatly improved if such an improvement is accomplished. Furthermore, we have a plan for measuring the frequency of initial positions that have no solution accurately. It is expected that such an initial position occurs once per several tens of thousands initial positions. It is easy to screen initial FreeCell positions for finding those that have no solutions even by using the current solver. However, a depth first search to an initial FreeCell position that the current solver cannot solve might require a huge hash table that cannot go in the main memory of work stations used in our experiments. If we have a small upper bound on the length of a shortest solution to an initial position of FreeCell, then we might be able to prevent the main memory from overflowing with the elements in the hash table. It is an theoretically interesting challenge to us to find an explicit value of the upper bound.

Acknowledgment

We appreciate kindly discussion by the members in the RIMS workshop. This work was supported by JSPS KAKENHI Grant Number JP15K00018.

References

[1] Achiya Elyasaf, Ami Hauptman, and Moshe Sipper. Ga-freecell: Evolving solvers for the game of freecell. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1931–1938. ACM, 2011.

[2] Achiya Elyasaf, Ami Hauptman, and Moshe Sipper. Evolutionary design of freecell solvers. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(4):270–281, 2012.

[3] FreecellGameSolutions.com©. Easy and difficult games — freecell solutions. `https://freecellgamesolutions.com/stats.html`. Referred on 20 May 2019.

[4] George T. Heineman. Algorithm to solve FreeCell solitaire games. `http://broadcast.oreilly.com/2009/01/january-column-graph-algorithm.html`, January 2009. The contents on 22 January 2009 obtained from the Internet Archive (`https://archive.org/`).

[5] Gerald Paul and Malte Helmert. Optimal Solitaire Game Solutions using A* Search and Deadlock Analysis. In *Ninth Annual Symposium on Combinatorial Search*, 2016.